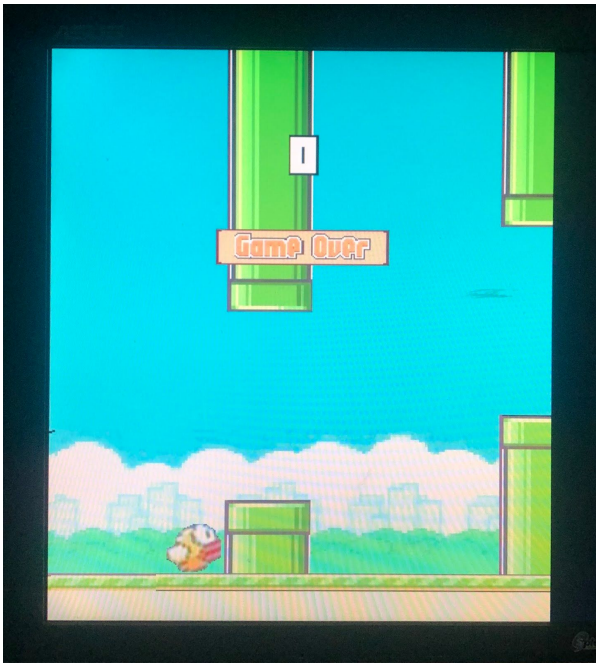
Final Project Report

一、Introduction&Specfications

Design Topic : Flappy Bird



Input/Output Table

Input	Function	Output	Function
Button BTND	Reset	VGA board	Display
Keyboard Button number5	Restart	7-segment display	Score
Keyboard Button number6	Bird spits fire	LED light LD0~LD3	Game level
Keyboard Button number8	Jump / Start		
Switch SW0~SW3	Frequency control		

Functions of the program:

- 用鍵盤的按鍵執行遊戲的指令。
  (1) 數字鍵盤5代表遊戲結束後重新回到一開始要開始遊戲的畫面。
  (2) 數字鍵盤6代表鳥噴火(當鳥噴火時，碰到水管不會死)。
  (3) 數字鍵盤8代表跳(如果在遊戲的開始畫面時按下8才會進行遊戲)。

- 遊戲分為四種難度，使用者可以藉調整switch來手動調整遊戲難度，目前的遊戲難度則會顯示在LED light上面。
- 7-segment display會顯示當前的分數(螢幕也會顯示分數)。
- 鳥碰到天花板、地板以及在沒有噴火的情況下碰到水管皆會死亡。

二、Motivation

在試玩過Flappy Bird之後，發覺網路上的版本難度太高，幾經數十次的不斷嘗試，分數仍舊是個位數，這樣的遊戲難度容易使玩家過於灰心，造成不良的遊戲體驗，因此，我想設計對於玩家難度適中的版本，讓許多對這類型有興趣的玩家能夠循序漸進，從引起興趣開始，依照自己的遊戲程度調控遊戲難度，可以由簡入難，讓遊戲更為豐富以及獲得更多成就感，使玩家獲得更棒的遊戲體驗。除此之外，在Lab6學到螢幕的概念有些鬆散，缺乏統整性，沒有機會系統性的整理相關的概念。對於以前沒有接觸設計過螢幕遊戲的我來說，是一個練習的良好機會，能夠自己重頭設計螢幕遊戲，從做中學習，不僅能更熟悉概念，也能和課程內容有觸類旁通的效果。

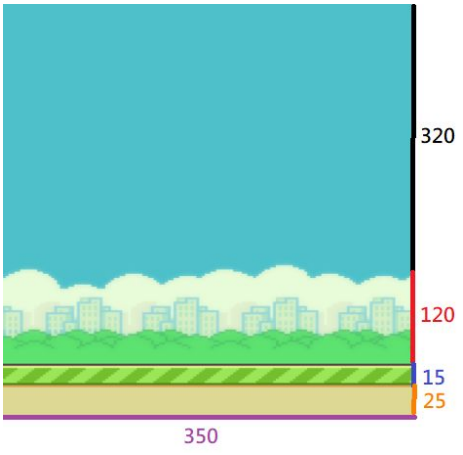
三、Designing the game

Process of designing the game:

(1) 讓水管能夠隨機改變高度並由螢幕的右方移動到左方。
(2) 做出鳥的彈跳功能。
(3) 取得水管的座標位置，判斷鳥collision的條件。
(4) 做出計分功能顯示在7-segment以及螢幕上。
(5) 做出鳥噴火功能。
(6) 做出整張的背景。
(7) 依優先度決定該如何顯示螢幕的顯示(screen coloring)。
(8) 設計遊戲的不同難度

圖片規格設定:
Pipes → 60x480
Bird → 40x40
Numbers(Score) → 300x40
Game Over → 120x30
Background → (blue sky → 350x320
cloud → 350x120
ground → 350x15
brown ground → 350x25)

Screen Displaying Range → 350x480



(一)Pipes(60x480)

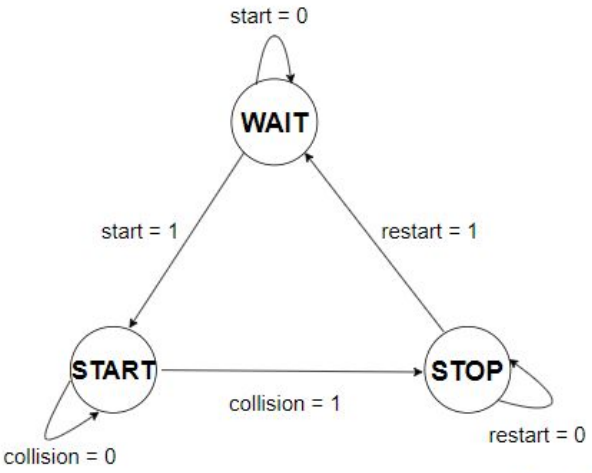
此部分實作在mem_addr_gen_Pipe.v的module中，以下為重要的參數：

input clk, rst;
input [9:0] h_cnt, v_cnt;
output [9:0] pipe1_x, up_pipe1_y, down_pipe1_y;
output [9:0] pipe2_x, up_pipe2_y, down_pipe2_y;
output [16:0] pipe_pixel_addr;
input start, restart, collision;

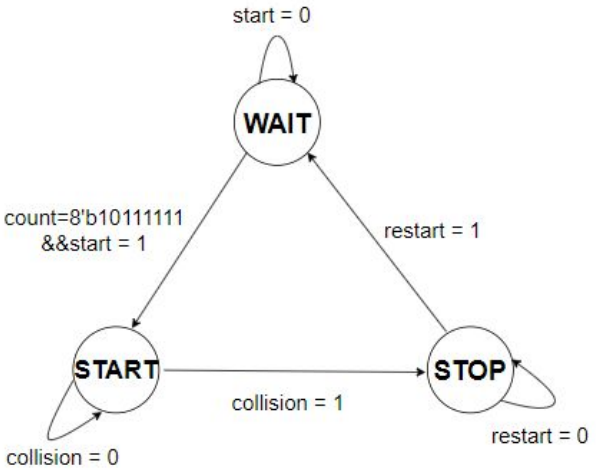
此部分為水管定義了三個state，分別為WAIT、STOP以及START。

因為在螢幕顯示上，一次最多會有兩根水管出現在螢幕上。一開始水管還尚未出現(WAIT state)，當按鍵8一按下(start=1)(START state)，第一根水管開始由螢幕h_cnt=470的位置由右方至左方移動出現時，**我們必須做出一個counter數到特定的數時(count = 8'b10111111維持住)，讓第二根水管接續出來**。而當水管由螢幕的右方移動到左方消失時，必須再從右方出現。
我們讓水管移動的方式是，鎖定螢幕範圍h_cnt <= pipe1_x && h_cnt + 60 > pipe1_x，而pipe1_x會隨著每一次的clk trigger，由470遞減至0，到0時又會回到470，重複不斷循環。如此一來就能讓水管移動，水管消失後又可以從右方出來。
(最後螢幕顯示的範圍是**350**x480，之所以讓水管由h_cnt=**470**的地方開始跑是因為要讓遊戲有**等待的感覺**)

而當水管移動的途中，鳥撞到了地板、天花板或是在未噴火的情況下撞到了水管，會有一個collision=1的信號讓水管停止(STOP state)，直到按下按鍵5(restart=1)，會讓遊戲回到一開始即將進行遊戲的畫面，水管則由STOP state回到WAIT state。
以下是兩根水管的State Transition Diagram：

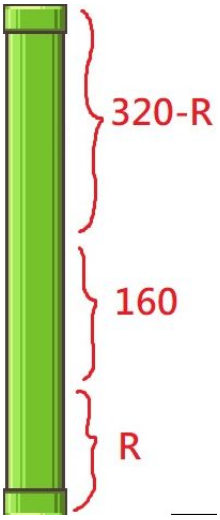


For Pipe1

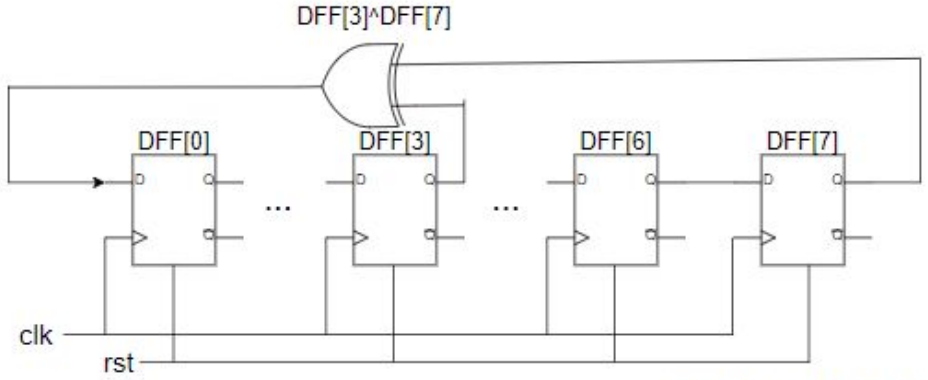


For Pipe2

在遊戲進行中，水管出現的高度是要隨機產生的，因此我們是利用之前在Lab3所學到的LFSR產生水管高度的隨機變數。如右圖，我們設定水管中間的間隙大小為60x160，間隙下方高度設為R，此部分是要顯示在螢幕下方；而間隙上方高度就是480-(160+R) = 320-R，此部份是要顯示在螢幕上方。由此可知道，我們所需R的範圍是介於0 < R < 320。因此我們需要設計出一個8-bit的LFSR產生這個範圍的隨機變數。



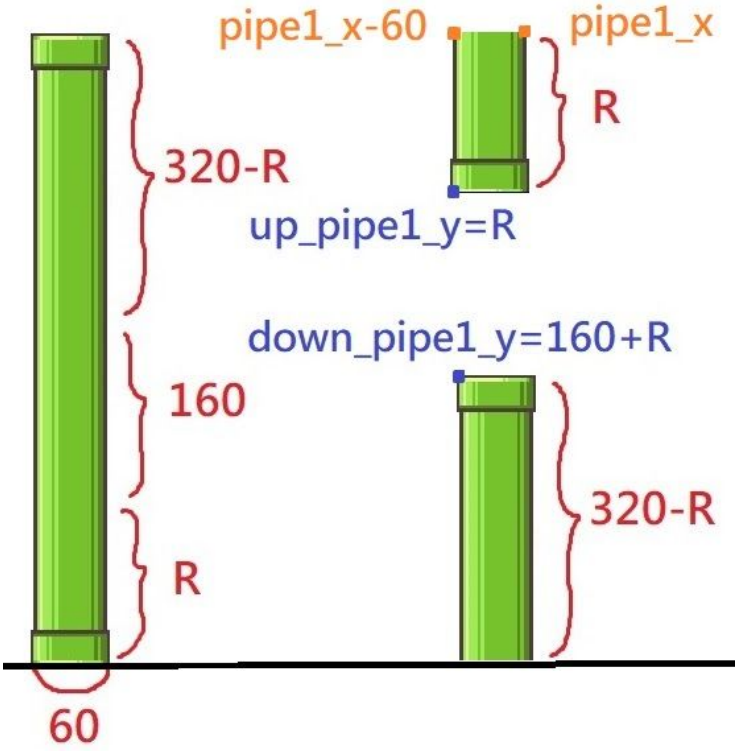
以下是LFSR的電路圖：



8-bit LFSR

以上的LFSR在reset時，DFF = 8'b10101101。

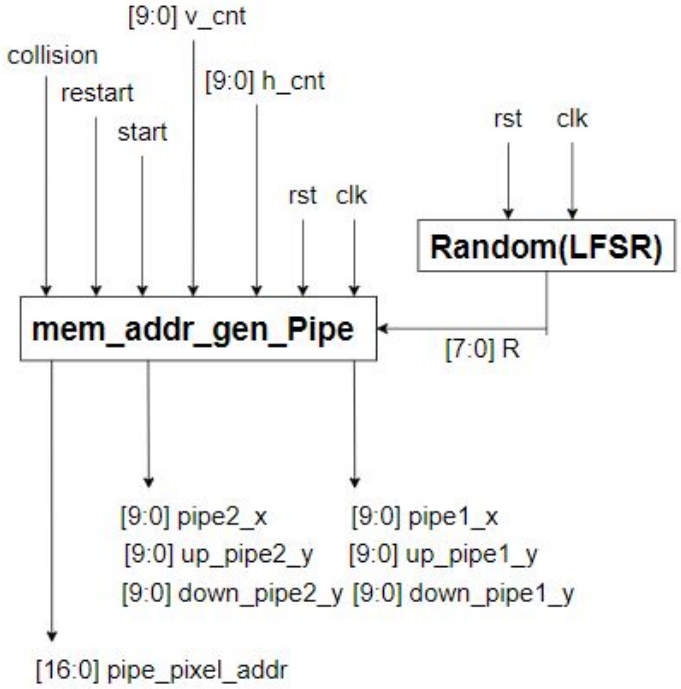
有了LFSR可以產生水管的隨機高度後，我們就有上下水管的y座標(上水管y=R,下水管y=160+R 如右圖)
這些水管的x,y座標需要當作output傳出去(之後螢幕填色會需要這些座標)



下一步是此部分最重要的，就是要寫入正確的水管圖片地址(pipe_pixel_addr)，這樣在水管的Block Memory中才能取出適當位置中正確的pixel。
舉上水管為例，上水管在螢幕顯示的位置範圍是鎖定在(h_cnt <= pipe1_x && h_cnt+60 > pipe1_x&& v_cnt > 0 && v_cnt < R)
，那麼pipe_pixel_addr = (v_cnt - (480-R))*60 + (h_cnt - (pipe1_x-60))
。

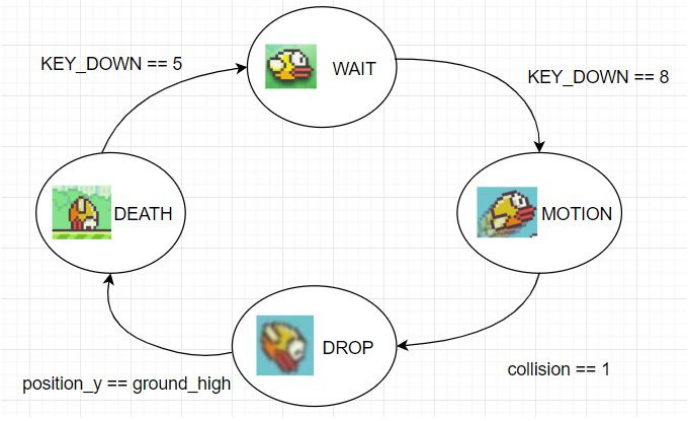
((v_cnt - (480-R))*60 + (h_cnt - (pipe1_x-60)是代表將原本未切割的水管取480-R<y<480的範圍，並要顯示在0<v_cnt<R(上水管的範圍)，所以v_cnt必須扣掉垂直位移量為480-R，並乘上圖片寬度60，再加上h_cnt減水平位移量(pipe1_x-60)，才能取到原圖適當的位置。)
其他水管沒有出現的範圍，我們將他的pipe_pixel_addr取0，也就是水管圖片中最左上方的位置，那個位置是深紫色(色碼為12'h534)的，所以水管沒有出現的範圍一律都是深紫色的。**為甚麼刻意讓水管的這張圖為深紫色的呢？原因在於之後可以方便又快速地判定鳥是否撞到水管。(在下方會談論到此部分)**

下方是此部分的Block Diagram：



(二)Bird

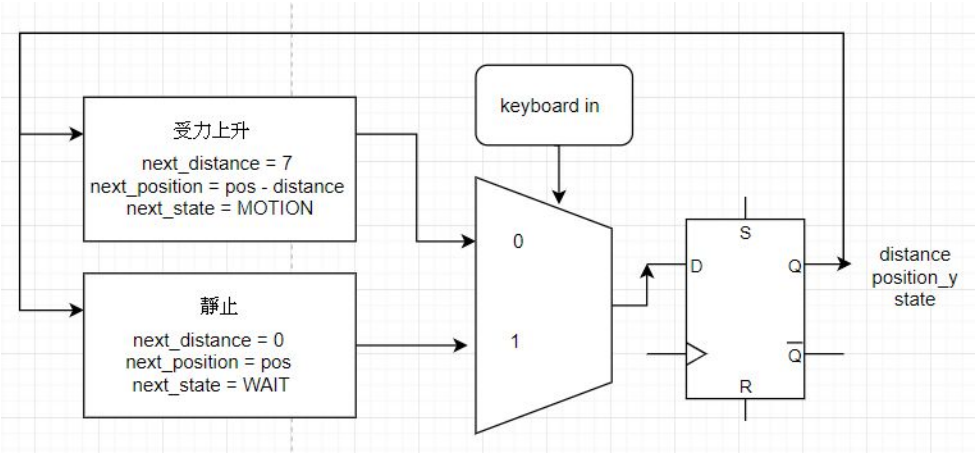
首先，我們定義鳥的state，以及鳥在每一個 state 中要有怎麼樣的行為。第一個 state 是 WAIT，在遊戲的一開始，鳥飛行在固定高度，當按下keyboard 按鍵 8 之後，會進到遊戲階段，即 state MOTION；在MOTION階段，當按下按鍵 8 時，鳥會向上彈起，若否，鳥會受地心引力作用往下墜，在遊戲階段中，鳥不能撞到天花板、地板、水管，若有衝撞情形發生，鳥會墜落，進到下一個階段DROP；在state DROP 中，鳥會由原來的高度等速下降，在墜落到地板之後，會進入到 state DEATH 中; 在 DEATH 階段，鳥會持續停留在地板直到玩家按下按鍵 5，按下按鍵 5之後則重新開始。以下為 STATE TRANSITION DIAGRAM。



在實作鳥的彈跳功能中，我們利用position_y控制鳥的高度，用distance來控制鳥在每個clk cycle 內的位移量。

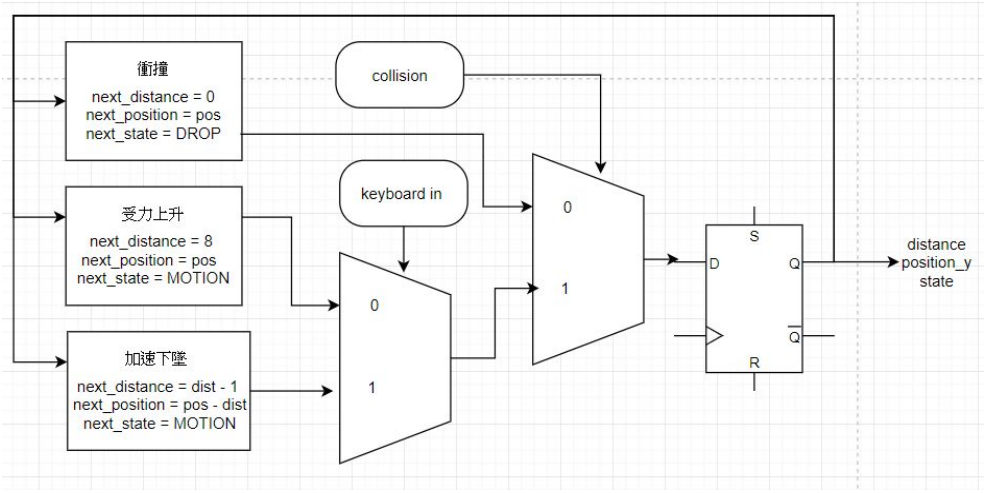
STATE WAIT

在此state中，透過一個簡單的小mux完成任務，當接受到keyboard 8 的 onepulse 訊號時，next_position_y 會 - 8（螢幕上移八格），distance 設為7，並進入到下一個 state，若否，鳥維持在同一個 state，next_position_y 等於 posiition_y 。下圖為此state 的邏輯圖。



STATE MOTION

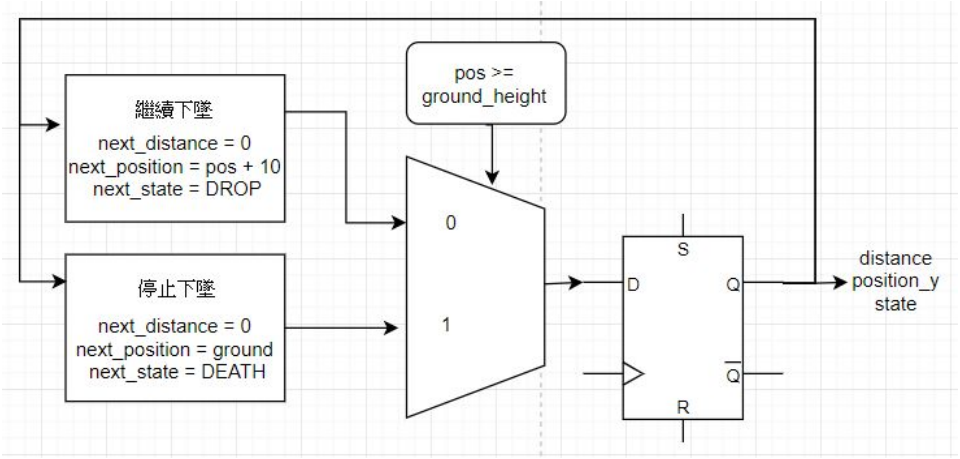
在此state中，當keyboard按下8，鳥會獲得一個初始distance值 8，而鳥的 next_position 則設定為 position_y - distance，意即鳥的座標上升 distance 單位， 而 distance 值每經過一個clk cycle 之後會 -1，因此鳥的座標上升的變化量會持續減少，當 distance 小於0時，鳥就會向下加速墜落。在此state 之中，若接收到collision 訊號，鳥則會進入到下一個墜落的state。下圖為 block diagram。



STATE DROP

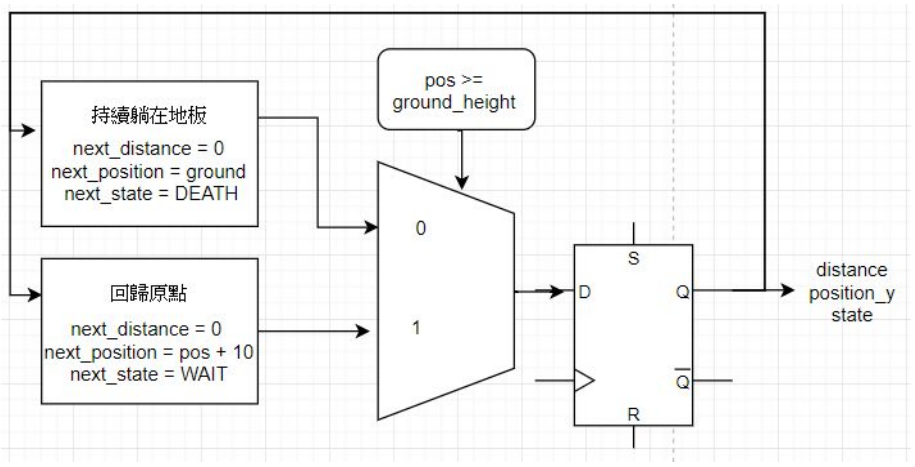
當衝撞情形發生進入此state 之後，鳥會等速下降直到撞到地板。在實作上是在將鳥的position值在一個clk cycle 內會加一個固定的值，直到鳥的高度降到地板的高度

為止，在position等於地板之後，鳥會進入到下一個 STATE。下圖為此state的邏輯圖。



STATE DEATH

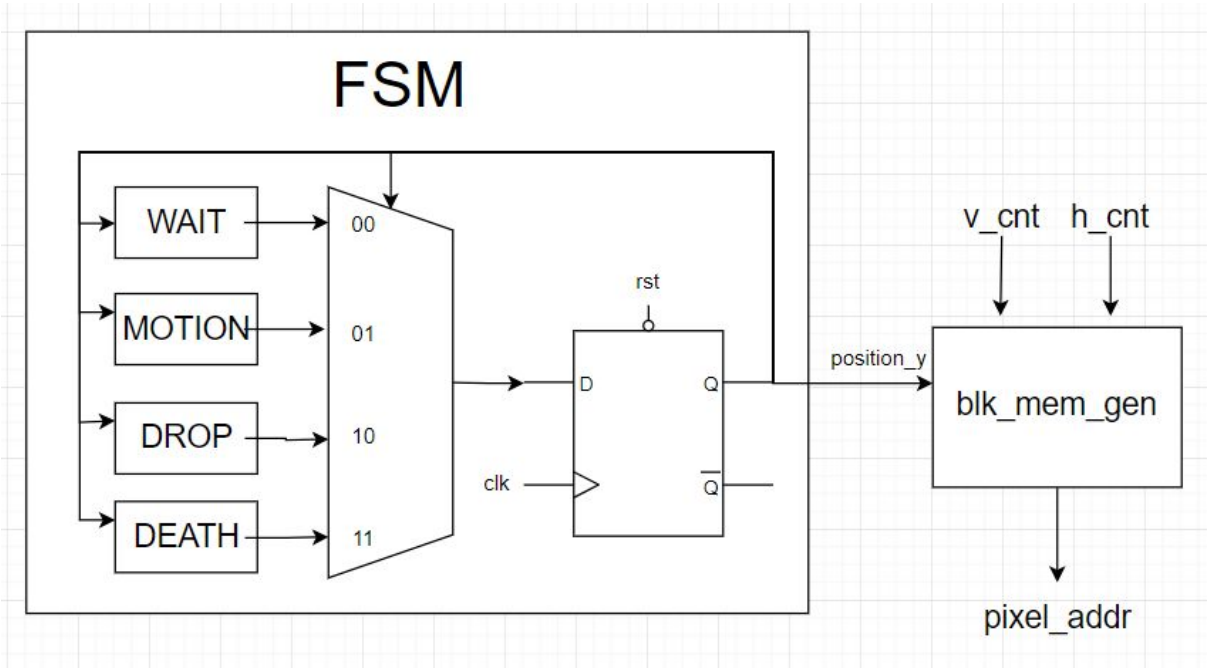
在此state中，鳥會持續停留在地板，若收到keyboard = 5 的onepulse 訊號，鳥會重新回到state WAIT。右上為此state的邏輯圖



鳥的圖片輸出

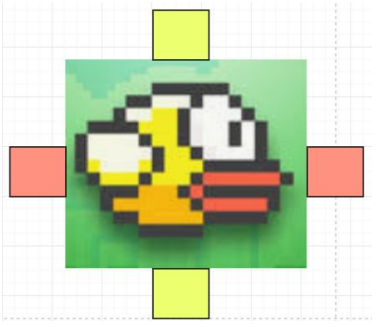
我們將鳥的中心點的 x 座標訂在 100， y 座標即為 position_y，而鳥的大小是 40 X 40，因此在掃圖檔色塊時需要將讀取的x座標的位置往左平移100，y座標的位置往上平移 position_y - 20，為了避免出錯及容易去背，當平移後在鳥圖片的邊界之外的pixel，統一給予鳥圖片中其中一個顏色為12'h0 的pixel_addr。

以下為此鳥的 block diagram



(三)Collision Detection

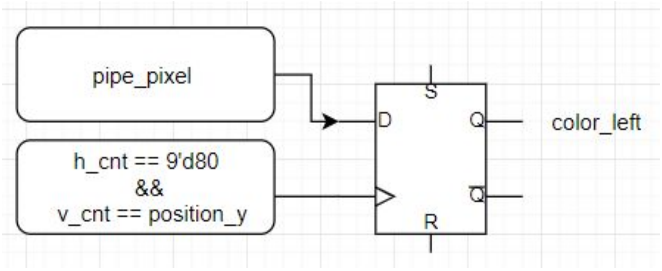
為了參數input output 的方便，我們直接將碰撞偵測的機制實作於mem_gen_addr_bird.v 的檔案中，與鳥的彈跳功能寫於同一個module。鳥的碰撞判定是以鳥的上、下、左、右作為偵測點，四個偵測點會去讀取水管底圖的色碼(註 1)，若取出之色碼屬於水管的色碼，就代表鳥的周圍與水管相連，因此就判定為衝撞，此外，上下的偵測點會額外偵測飛行高度是否合法，若高度超過天花板或低於地板，一樣判定為衝撞。以下為色塊偵測示意圖。



接著我們要來探討如何讀取、儲存色碼，以及如何用色碼判定。螢幕顯示圖片的方式是分別將一格一格的pixel 顏色顯示出來，當h_cnt 與 v_cnt 數到該格的位置時，會將該格欲顯示的RGB值輸出，而我們要做的事，就是分析 h_cnt 與 v_cnt 數到探測點時所發生的事。

在h_cnt 及 v_cnt數到探測點之後，我們必須將那個時刻所輸出的水管顏色記錄下來，為了使值能夠延長到一張圖片輸出的週期，我們需要用到DFF；以左偵測點為例，我們先宣告一個signal_left, 當h_cnt v_cnt數到左偵測點的時候，signal_left 設為1，其餘時刻設為0，而我們將此訊號作為posedge trigger 的訊號，用以讀取左邊點的

水管底圖顏色值，由於在一個圖片週期內，signal_left 為 1 的次數只有一次，因此能夠確保能夠儲存探測點的顏色值到下一個圖片週期。以下為
示意邏輯電路圖。



同理，上下右的顏色也能夠以此方式記錄下來。下一步，我們判斷已取得之上下左右顏色是否為非水管區域(註)，若是，遊戲繼續進行，若否則判斷為collision；再結合上下的邊界判定，collision的式子能夠寫為：

collision = ((color_up != 12'h534) || (color_down != 12'h534) || (color_left != 12'h534) || (color_right != 12'h534) || (postion_y < 5'd20) || position_y > 9'd419)

<ps> 非水管區域，在製作水管區域底圖時，在水管沒覆蓋到的區域，我們統一將該區域的色碼設為 12'h534，因此在做色碼判斷時，若取得的值等於12'h534，則判斷鳥是在非水管區域。

(四)Score Board

記分板的計分機制主要是看兩根水管的x座標是否等於鳥的x座標(鳥的x座標其實不會動，會一直維持在螢幕80<h_cnt<120的位置，鳥只會垂直上下移動)。
此部分分成兩個部分，第一部分為記錄分數；第二部分為顯示。而顯示又分成兩個部分，一為七段顯示器上的顯示；二為螢幕顯示。

(1)紀錄分數

紀錄分數的部分寫在mem_addr_gen_Pipe.v的module中(為output)。
以下會有三個數紀錄分數的個位數字、十位數字以及百位數字，分別為[3:0]score0, score1, score2。在遊戲開始前，會將所有位數歸零。
遊戲開始時，當偵測到水管的x座標等於鳥前端的x座標時(pipe1_x == 10'd120 || pipe2_x == 10'd120)，當score0!=4'd9時，score0會加一次1'b1；而當score0=4'd9&&score1!=4'd9，score0會歸零、score1會加一次1'b1。score2以此類推。此紀錄分數的部分只是判斷數字什麼時候該進位，在之前Lab4用FPGA板所實作過的stopwatch的概念相同，在此不贅述。

(2)顯示

a.七段顯示器

```
always @(*) begin
    case(stcl)
        2'd0: seg = seg0;
        2'd1: seg = seg1;
        2'd2: seg = seg2;
        2'd3: seg = seg0;
```

此部分實作在seven_seg.v的module中，以下為重要參數：

```
input [1:0] stcl;
input [3:0] score0, score1, score2;
output [6:0] seg;
output [3:0] an;
reg [6:0] seg0, seg1, seg2;
```

此部分按照先前幾次Lab所操作的手法相同，分別將各個位數的不同數字分別給好七段顯示器中該亮哪幾條線，在分別給對應的暫存seg(seg0,seg1,seg2)，最後依人眼看不見的頻率的訊號(stcl={clk_15,clk_14})依序顯示不同位數的數字，就能完成實作。如右圖。

b.螢幕顯示

在此有用了一張大小為400x30的數字圖片，數字跟數字間的水平距離差不多是40個pixel(包含空隙)，如下圖：

此部分實作在mem_addr_gen_num0以及mem_addr_gen_num1的module中(沒有mem_addr_gen_num2是考慮到遊戲難度，要到達100分以上有很高的難度以及技術，因此螢幕顯示的部分沒有做第三位數字)，以下為重要的參數：

```
module mem_addr_gen_num0(
        input [9:0] h_cnt, v_cnt,
        input [3:0] score0;
        output [16:0] num0_pixel_addr);

module mem_addr_gen_num1(
        input [9:0] h_cnt, v_cnt,
        input [3:0] score1;
        output [16:0] num1_pixel_addr);
```

score0會大致顯示在螢幕的160<h_cnt<185,70<v_cnt<105的範圍；score1會大致顯示在螢幕的130<h_cnt<155,70<v_cnt<105的範圍。根據score0以及score1產生的數字，num0_pixel_addr以及num1_pixel_addr需要產生對應數字圖片的pixel address。以下會敘述我們用過怎麼樣的方法去圍出各個數字的相對位置。

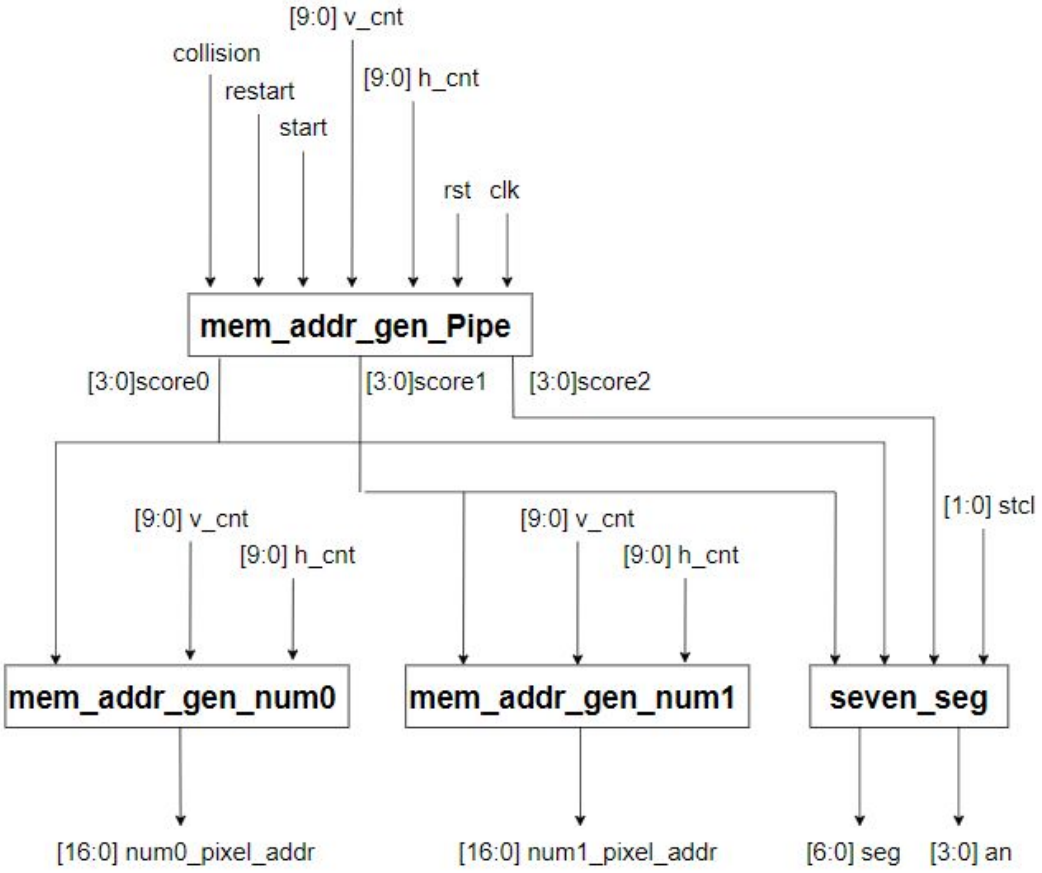
一開始我們有想到很快速的方法，可以簡單快速地讓數字顯示在螢幕上，想法如下：我將數字外圍的顏色用小畫家塗成純紅色，因為每個數字之間的距離差不多是40個pixel，所以我們圈選數字的範圍是以40的倍數來取，之後在螢幕填色時，如果**要顯示此部分的數字圖片時，就指定這個範圍的紅色(在.coe檔中可以知道紅色的色碼)要以背景取代**，這麼一來，有些區塊就不會混雜其他圖片的顏色。如下圖的coding

```
end else if(h_cnt >= 130 && h_cnt < 190 && v_cnt >= 70 && v_cnt < 110) begin
    if(score_pixel == 12'he12) 紅色的色碼
        {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? combined0_bg_pixel:12'h0;
    else                                                背景的pixel
        {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? score_pixel:12'h0;
```

但是很不幸的，好像用小畫家填色，他的紅色非單一的純色(色碼亂掉)，所以在指定色碼的時候，還有其他色碼的紅色顯示在螢幕上，導致螢幕有類似染色的現象。這個是我們所遇到的困難之一。

最後只好使用最傳統暴力的方法：**一個數字一個數字慢慢取其範圍**。在取數字範圍中，最麻煩的是像4或是7這兩個數字，因為這兩個數字要拆開取範圍；至於其他數字，因為都是長方形，所以算比較好處理。num0_pixel_addr即會等於score0的數字所對應的螢幕圖片pixel address；num1_pixel_addr即會等於score1的數字所對應的螢幕圖片pixel address。

下方是此部分的Block Diagram：

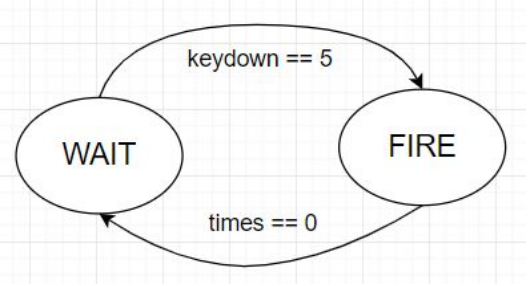


(五)Bird Spitting Fire

Input clk, rst, h_cnt, v_cnt, position_y, gogacu(keyboard_in);
Output pixel_addr, gogacu_signal;

火球的finite state machine

為了使平日時待機，按下按鍵能夠噴火，我們使用了兩個state及一個register (gogacu_signal)來完成功能。在遊戲的一開始，火球所在的state 是 WAIT，在此state 時，gogacu_signal 為0，意即無法噴火；當接收到 keydown = 6 後，state會切換到 FIRE，在此階段，gogacu_signal 會變為1，代表可以噴火，進入此state後，會啟動一個register (times)倒數，當 times 從 20 倒數到 0 時，state 會切回到 WAIT。以下為state transition diagram。



火的圖像輸出

若 gogacu_signal == 1 時，代表鳥可以噴火，因此需要火的圖象輸出。火的範圍設定是依據鳥的位置，火的上界是position_y - 56，下界是position_y + 56，左邊界是 120，右邊界是320，因此水平掃描的位置要左移120，垂直掃描位置要上移56，因此讀取時的式子可以寫為：

pixel_addr = (200 * (v_cnt - (position_y - 56)) + h_cnt - 9'd120

為了取值不會出錯及去背考量，平移之後在火球圖外的pixel值一律設置為 12'h0(黑幕)(RGB值為0的pixel_addr)，因次判斷是否在範圍內的式子可以寫為：

h_cnt >= 120 && h_cnt < 320 && v_cnt >= (position_y - 56) && v_cnt < (position_y + 56)

若 gogacu_signal 為 0 時，帶表火不需要噴出，因此output 直接為12'h0，此外，為了讓火球看起來更炫炮，我們將條件判斷式改為在符合上述條件之後，仍須符合 times 為奇數的條件才能輸出火球，如此一來火球就可以一個週期亮一個週期不亮，達到閃爍的效果。

噴火期間的免死功能

當gogacu_signal 為1時代表鳥正在噴火，因此在此state 時鳥碰到柱子不會死亡，為了達成此效果，我們將先前所解釋過的鳥的 state motion 稍作修改，我們將原死亡的判斷式 collision 改為 collision && (~gogacu),

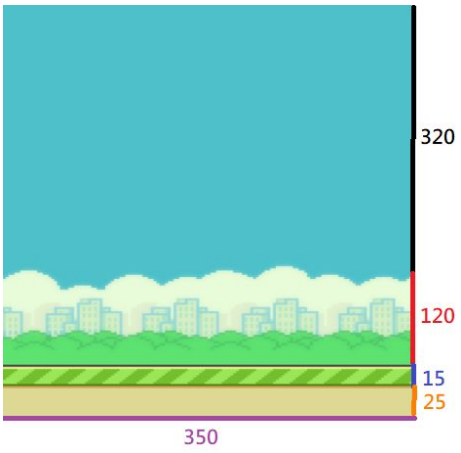
(六)Background

此部分實作在mem_addr_gen_Ground_Cloud以及pixel_gen_Background的module中，以下為較為重要的參數

```
mem_addr_gen_Ground_Cloud (
        input [9:0] h_cnt,v_cnt,
    output [16:0]ground_pixel_addr,
                        cloud_pixel_addr);
```

```
pixel_gen_Background (
    input [9:0] h_cnt, v_cnt,
    input [11:0] ground_pixel, cloud_pixel,
    output [11:0] background_pixel);
```

關於背景這塊，因為一張完整背景的圖片所需要的像素太大(350x480=168000，超過FPGA板所能容納的記憶體)，所以我們將背景切成4個區域，分別為blue sky → 350x320、cloud → 350x120、ground → 350x15以及brown ground → 350x25，如右圖所示。



因為blue sky以及brown ground是屬於單一的純色，因此我們可以直接assign色碼至此區域填色，不需要此部分的圖片。(blue sky色碼 → 12'h4cc, brown ground色碼 → 12'hdd9)，填色部分是在pixel_gen_Background的module中實現。以下說明blue sky以及brown ground是如何填色

```
if(v_cnt < 320 && h_cnt < 350)
        background_pixel = 12'h4cc;
else if(v_cnt > 455 && h_cnt < 350)
        background_pixel = 12'hdd9;
```

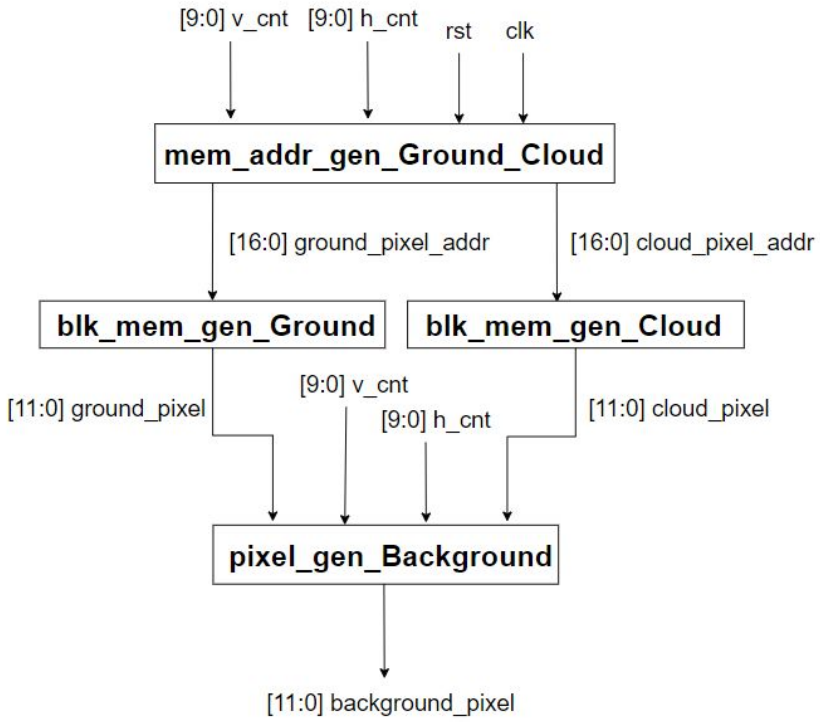
關於ground的部分我們只需要350x15大小的圖片，顯示在螢幕0<h_cnt<350, 440<v_cnt<455範圍的位置，並且讓它由右往左移動。在遊戲結束的時候，地板會接收到一個gameover=1的訊號(這個訊號跟collision=1類似)，此時要馬上停止移動。地板移動的功能大致上跟水管移動的概念相同，在此不贅述。

至於cloud的部分，會顯示在螢幕0<h_cnt<350, 320<v_cnt<440範圍的位置。然而，因為350x120(=42000)的圖片大小還是太大，所以我們選擇採用**壓縮圖片**的方式降低圖片的畫質，所以在外觀上如果「仔細觀察」，會發現中間會有類似鋸齒狀的情況產生，為了做出完整的遊戲，這個部分無法避免。壓縮圖片的做法是，將**原本為350x120的圖片，轉成175x60大小的.coe圖檔**(大小跟原本的350x120差4倍)，接著用h_cnt以及v_cnt**往右移一位**(除二)的方式，寫進cloud_pixel_addr中，如此一來，原本圖片1個pixel就能在螢幕上顯示成4個pixel的大小。因此此部分的式子可以寫成以下：

```
        if(v_cnt >= 320 && v_cnt < 440 && h_cnt < 350)
         cloud_pixel_addr = (((v_cnt>>1) - 160) * 175 - 1 + (h_cnt>>1));
```

當cloud以及ground獲得圖片正確的pixel address(ground_pixel_addr, cloud_pixel_addr)後，他們會**分別去對應圖片的Block Memory(ground → blk_mem_gen_Ground、 cloud → blk_mem_gen_cloud)找正確位置的pixel。**獲得正確的pixel後，會統一去pixel_gen_Background的module中填特定範圍中所對應的pixel(包含blue sky以及brown ground填色碼)。

以下是此部分的Block Diagram：



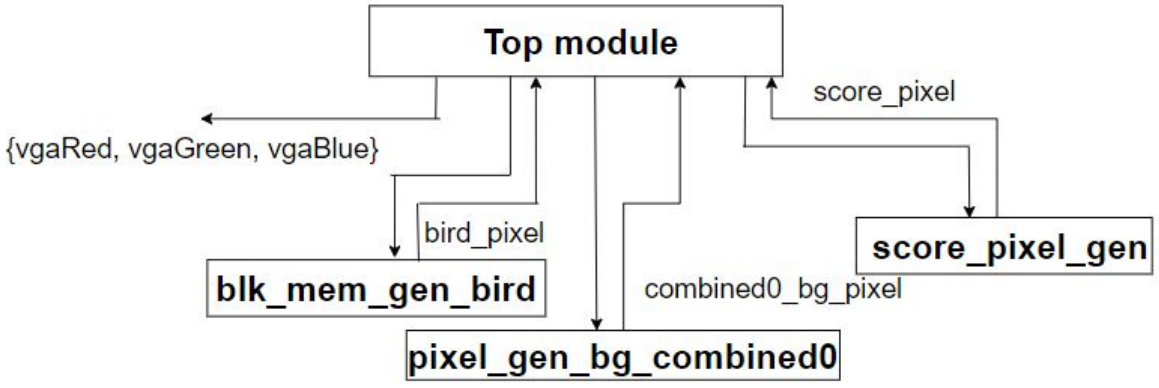
(七)Screen Coloring

最後螢幕填色的關鍵，是依照圖片顯示的優先度先後填色，假設今天要填鳥的顏色，那麼第一步就是先鎖定螢幕的範圍，然後在指定鳥的pixel給此範圍，即完成此範圍的填色。以下的code為舉例：

if(h_cnt >= 80 && h_cnt < 120 && v_cnt >= ((bird_position_y - 20) % 480) && v_cnt < ((bird_position_y + 20) % 480))

{vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? bird_pixel:12'h0;

此部分實作在在top module中。以下為此部分的Block Diagram：



(pixel_gen_bg_combined0為合成background_pixel以及pipe_pixel的module)

(八)Game Level

此部分只要設計一個always block去看兩種情況，第一種是遊戲的分數的多寡，分數越高，給的clk頻率就要越快；第二是手動調整switch控制遊戲難度的部分，如果相對應的switch被扳成1'b1，那麼對應的遊戲難度的clk就會被賦予。最後在將對應遊戲難度輸出到LED lights上([3:0] game_level)。(難度一：game_level=4'b0001；難度二：game_level=4'b0011；...)

四、Experimental Results

整體遊戲時做出來之後，有達到預期之結果，在流暢度、擬真度、靈敏度上皆有達到預期，以下將逐一討論此遊戲內各部分的實驗結果。

(1) 水管；使用LFSR來當作隨機高度的產生器，玩家在玩遊戲不易遇到重複高度的水管，水管在截切面及邊界的顯示正常，不會有不連續的情形。

(2) 鳥：鳥的物理公式採用等加速度向下的寫法，在每個clk墜落的距離與時間成正比，在螢幕display時也能使玩家感受出鳥有加速下降趨勢；此外，在keyboard的input到鳥的彈跳的時間差極短，在畫面的感受上是同時進行的。

(3) 碰撞判定：碰撞的判定是依據鳥的上下左右的四個偵測點有沒有和水管色塊重疊，在實際運行中，若是鳥的身體有部分和水管碰到但沒碰到偵測點，鳥一樣可以通過水管，但偵測點排列緊密，玩家要觀察力很敏銳才能觀察出。在運行階段鳥的碰撞看起來合理，不會有撞到水管不掉，或不撞到水管掉下來的情況發生。

(4) 分數顯示：一開始顯示時由於背景紅幕的色碼在轉檔案時會有些許的不同，每一個pixel的RGB值會稍微跑掉一些，因此在去除紅幕時會有一些殘餘色碼沒轉換到，再顯示上會有混雜離散的小色塊，之後所採取的方式是採用邊界判定，採用此方法之後的顯示就變為正常。

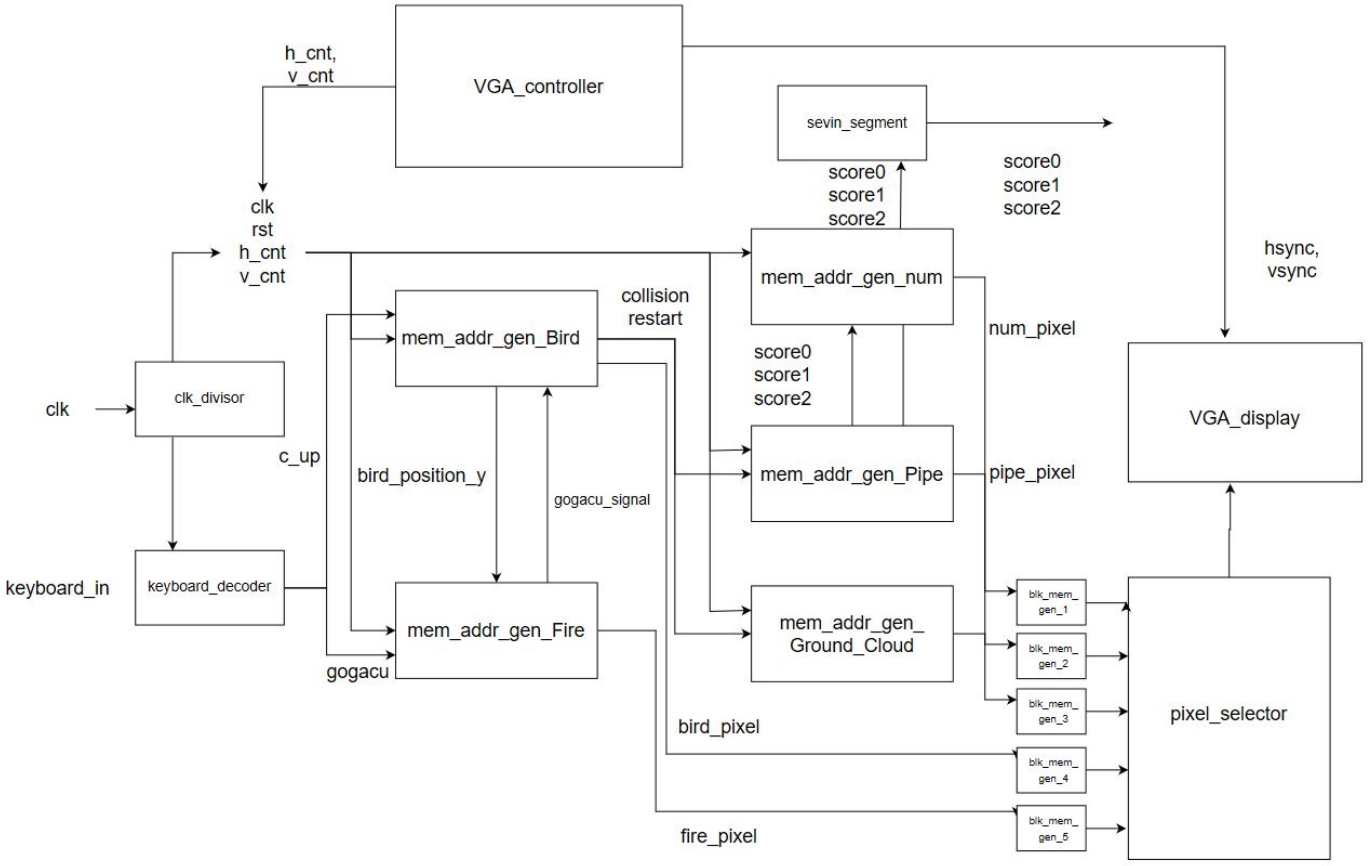
(5) 噴火特效：火球的顯示一如預期，且圖片的clk正好與火球所需的閃爍頻率差不多，顯示上流暢。

(6) 背景圖片：一開始選取的背景圖片畫值較高，但由於畫質高的圖片一下子就將記憶體空間占滿，因此我們改由存取較低畫質的圖片然後將其放映到整個空間，因此在顯示上會看到有一些區域會有不連續的色塊，有一些區域較為模糊。

(7) 疊圖：所有的特效顯示是依照圖片放映的優先權而寫的條件判斷式，在長方形圖片上採用的是邊界判定，因此在顯示上較為正常；但火球、鳥...等一些圖片的形狀為不規則形，因此這類圖形經過處理過，我們會先用長方形圖檔存，然後再將目標之外的顏色塗上單一顏色，使用條件判斷把顯示該顏色的地方去掉，但在圖片的邊界上著色會有困難度，有些地方會沒上到色，有些地方將原

圖塗掉使形狀稍微失真，因此在不規則圖形邊界上若仔細看會發現有一點雜色顯示出。

以下為這個遊戲大致上簡略的Block Diagram：



- <ps>每個pixel_addr都要去相對應的blk_mem_gen去取對應的pixel，最後回到top module中填色輸出{vgaRed,vgaGreen,vgaBlue}，完成螢幕輸出。

五、心得

經過最後的final project，真的收穫到不少東西，像是對螢幕如何輸出這一塊有很大的進步、該如何跟組員分工才是最有效的分工、怎麼樣的coding style會讓彼此更進入狀況以及debug的方法。我們這組開始做final project的時間有點晚，因為彼此在那段時間都有很多考試，但是還好有一個禮拜我們有比較多的時間可以做final project，回憶起那個禮拜，我們從早到晚都在資電館一起討論、一起鑽研、一起分工打code，慢慢一點一滴的累積。原本螢幕上只有水管移動，其他全部都是黑屏，漸漸地遊戲中有鳥的加入、有背景的加入，最後成功地讓分數顯示在螢幕上，回憶到此，仍覺得有很大的成就感。真的很感謝邏輯設計實驗的教授以及認真辛苦的助教們，在每次的Lab都給我們這麼多回饋，也常常關心我們的狀況，我真的覺得很慶幸能夠修到這門課，我

想這門課是我從上大學以來，修過最有價值以及學到最多東西的一門課程，再次感謝教授以及所有助教們，你們真的辛苦了！